End of Result Set

```
L1: Entry 1 of 1                    File: TDBD                    Nov 1, 1997
```

TDB-ACC-NO: NN9711117

DISCLOSURE TITLE: Efficient Logging of Transactions on Persistent Information in General and Databases in Particular

PUBLICATION-DATA:
IBM Technical Disclosure Bulletin, November 1997, US

VOLUME NUMBER: 40
ISSUE NUMBER: 11
PAGE NUMBER: 117 - 120

PUBLICATION-DATE: November 1, 1997 (19971101)

CROSS REFERENCE: 0018-8689-40-11-117

DISCLOSURE TEXT:

Disclosed is a methodology to log changes to persistent information. The described methodology applies when the persistent information is small enough to fit in main memory. The main benefits of this methodology are simplicity and efficiency. Assuming that the database is initially in a consistent state, a sequence of changes to the database that end in another consistent state is usually called a transaction. It is generally agreed that transactions have four properties called Atomicity, Consistency, Isolation, and Durability (ACID). This methodology deals with the first and last of those properties. The property that a transaction either succeeds completely (is committed) or is completely rolled back is called Atomicity. The property that once a transaction is successfully committed it survives system failures is called Durability. Durability is generally achieved by means of recording changes to disk. The current methodology is based on the following two assumptions: 1. The database is small. The entire contents of the database can reside in virtual memory (not necessarily main memory). 2. The underlying (Operating) System provides facilities for opening an existing or new file, sequentially reading from the file, sequentially writing to the file, forcing the file to persistent store, either removing a file or renaming a file.

A general description is provided first, followed by a specific preferred embodiment of this invention. File format: The file that contains the persistent data is made up of two parts: 1. The snapshot which is a streamed representation of the persistent data which appears at the beginning of the file and is either terminated by a special symbol or is length-encoded. 2. The log which is made up of a sequence of zero or more descriptions of operations on the persistent data. Each operation is either terminated by a special symbol or is length encoded. An operation is preferably a description of a transaction on the data. A transaction is, among other things, an operation that should either occur completely or not occur at all (atomic). Overall process: When the program starts its operations, it recovers the file that contains the persistent data (see recovery below). As the program executes, all changes to the persistent data (operations) are appended to the file in some encoding that can be used to replay the operation.

Cleanup: To avoid having the file grow without bound, a cleanup routine executes periodically based on some criteria such as log size, ratio of log size to data size, number of operations in the log, system activity, etc. Optionally, the cleanup routine applies a sanity check to the in-memory data structures that will be streamed out. If such a check fails, there is a software error. In this case, the program has several choices: 1. terminate execution (failfast approach). 2.

terminate and restart. This approach is used in the hope that the software error was a heisenbug, that is, that the bug is not reproducible. The program restarts and recreates the in-memory image of the persistent data from the file. 3. repopulate the in-memory image from the file. This approach offers faster recovery times than 2. but may fail to correct the problem if the problem is, for example, a bad pointer in the non-persistent data structures. The cleanup routine writes a snapshot of the persistent data to a new file (the backup file) whose name is composed by applying a transformation to the original file name. Then the backup file is atomically renamed, if possible, to replace the original file. If atomic renames are not supported by the host operating system, the original file is first removed and the backup file is then renamed. Immediately after the cleanup process finishes, the log portion of the file is empty. Recovery: Recovery is performed when the program first accesses the file. This is done in three steps: 1. The file is opened. If the host operating system does not provide atomic rename operations, the open may fail. This would happen if the system crashes during cleanup after the original file is deleted and before the backup file is renamed. In this case, the recovery routine renames the backup file to the original file name and tries to open again. To do this, the name of the file is transformed using the same algorithm used by the cleanup routine. 2. The snapshot is read to create an in-memory image of the persistent data. 3. The log is read and replayed. Each operation in the log is sequentially read and executed. Two special cases exist: a. If an invalid operation is found, it and subsequent operations are ignored. Finding an invalid operation usually indicates a programming or hardware error. By not replaying those operations at least the persistent data is maintained in a non-corrupted, consistent state. b. A truncated operation description is not replayed during recovery. Such a truncated operation was probably the last attempt to modify the state of the system before a software or hardware failure and is considered not committed. In a preferred embodiment of this methodology, the persistent information is a database of configuration information. Such databases are usually quite small, must be persistent, and their correctness is critical to the correct function of the system that relies on said configuration. The following is a description of our preferred embodiment of this invention: The persistent data is maintained as a relational database. The relational database is represented in the flat file as follows (in BNF notation): ::= ::= { ::= ; } ::= = ::= ; ; and are SQL identifiers. is the ASCII encoding of the value of a column. We allow the values null, true and false, integers represented as a sequence of decimal digits, and strings surrounded by single quotes.

As an example, here is the representation of a small database: persons { person = 'Joe' age = 29 insured = true lives_in = 'Austin'; person = 'John' age = 30 insured = false lives_in = Paris';

} cities { city = 'Austin' country = 'USA'; name = 'Paris' country = France'; } ; This database has two tables, each with two rows. In this case, the log is empty, but when the configuration database is modified (via Structured Query Language (SQL) statements), those statements are placed in the log portion of the file. Each statement is terminated with a semicolon. The SQL statement and the terminating semicolon are written to disk before the database returns control to the caller. So all operations on the database always append to the file, they never attempt to modify the information stored in the tables position of the file. If the system crashes while a write is taking place, the database system does not consider an SQL statement valid unless it is terminated by a semicolon, so truncated statements are not played back during recovery. In order to recover the space used by the log, a clean up routine is executed periodically to clean up the log. This routine is executed whenever the log reaches a certain threshold. This routine can be externalized in the form of an Application Program Interface (API), as well as a command to allow users to cleanup the database (maybe so that it can be edited). The clean up routine performs the following operations: o Write out a snapshot of the current contents of the database to a backup file whose name is composed by applying a transformation to the name of the original file. This backup file now has the contents of the database as a sequence of table descriptions terminated by a semicolon. The log is empty as it has already been applied to the database. Whenever the backup file and the original file exist, either they represent identical databases, or the backup file is not complete (is not terminated by a semicolon.) o Atomically change the name of the backup file to the original file. This replaces the original file with a file that contains an empty log. If the operating system does not have an atomic operation to rename a file, this can be done by first removing the original file and then renaming the backup file. This introduces a window where the original file does not exist. If the machine were to crash at this point, the configuration database would not be found. In this case, the recovery routines in the database manager apply the same transformation to the original file name, rename the backup file appropriately and try again. The grammar for the database and log outlined above are meant only as an example. Likewise, the references to SQL and relational databases are provided only for explanatory purposes. The approach outlined here works with any persistent data to which operations can be applied. The file need not be encoded in ASCII; any other encoding will work just as well. This approach works very well with small databases, such as those that maintain configuration information. These databases can be read and processed completely in memory and only have to be streamed out to disk to reclaim the space occupied by the log. When operations are performed, they are processed in memory and the only Input/Output (I/O) used is to append a description of the operation that was applied to the database. This is the minimum amount of I/O needed to commit the operation to disk. In this sense, this solution is optimal.